DESIGNING AN OBJECT-ORIENTED MATCHING TOOL

Hardo Müller
Institut für Photogrammetrie
Universität Bonn
Nußalle. 15, 53115 Bonn, Germany
Ph.: +49-228-73-2721, Fax: +49-228-73-2712
E-mail: hardo@ipb.uni-bonn.de

KEY WORDS: Design Pattern, Unified Modeling Language, Matching Technique, Feature Vector

ABSTRACT

A semiautomatic building extraction system has been extended by an automatic matching tool. It is used for an automatic measurement of building-heights and a semiautomatic determination of ground-heights. The object-oriented design of this matching tool gives the motivation for a design pattern of a general matching tool. This design pattern describes the object-oriented design of implementing several matching techniques within one system. It is able to be applied to several kinds of topographic objects, if a matching technique is known. We show as an example the implementation of a point matching tool, which uses the matching techniques Intensity Correlation and Gradient Correlation as a refinement of Feature Vector Matching.

1 INTRODUCTION

A semiautomatic building extraction system is being developed to extract 3D-vector data from aerial images (Lang and Schickler, 1993, Englert and Gülch, 1996, Gülch and Müller, 1997, Gülch, 1997)[1]. The semiautomatic system is based on interactive form and pose adaptation of building primitives. Due to an object-oriented system design it is easy to extend the system with new automated tools. Automation is necessary to accelerate the acquisition process.

Automation has so far been successfully applied for user-guidance, extraction of the texture or the docking of primitives for generation of complex buildings. Single primitives can be matched automatically using robust pose clustering methods.

A new automation tool is a general matching tool, which is able to yield 3D-descriptions from a given set of 2D-images. Actually it is realized as a point matching tool, intended a) to measure homologous points of the buildings b) to measure homologous points nearby on the ground to derive the height of the ground plan of the building and c) to measure other features like road boundaries.

Stereo matching is a well known problem and several solutions and application examples can be found in the literature (Förstner, 1986, Straub, 1991, Heipke et al., 1997). We want to integrate the proved matching techniques in an object-oriented design, to emphasize on the advantages of object-oriented modeling in this context. The object-oriented design of a matching tool offers the possibility

- to make use of the already implemented class interface (Gülch and Müller, 1997),
- to implement several matching techniques, using common templates,
- to use generic programming techniques for implementing algorithms on an abstract level,
- to hide the implemented matching technique, which will reduce dependencies in the software design,
- and to reuse the module within another context.

We want to depict this design as a design pattern (cf. section 3), which describes a recurrent software design problem and a generic schema of solution (Buschmann, 1996). This pattern is not only available for point-objects, but also for other objects, like edges or more complex topographic objects.

We use the Unified Modeling Language (UML), which is described in (Rat, 1997b), for a graphic representation of our dynamic and static object-model. The UML has adopted elements of the OMT (Rumbaugh et al., 1991) and the Booch-method (Booch, 1994). It is proposed for a standard at the Object Management Group (OMG).

Section 2 gives a short introduction into object-oriented modeling, UML, and Design Patterns. Especially the description-schema of a design pattern according to (Gamma et al., 1995) will be explained in section 2.3. The actual Design Pattern "*Matching Tool*" is presented in section 3. We finish this paper with some conclusions in section 4.

2 OBJECT-ORIENTATION,UNIFIED MODELING LANGUAGE AND DESIGN PATTERNS

This section gives a short introduction into object-orientation, by defining some of the major expressions relevant here, and describes the Unified Modeling Language (UML), which we use in our notation.

2.1 Object-orientation

Definitions of "Object-oriented programming" and "Object-oriented design" are described in (Booch, 1994):

[1] This research is supported by BMBF/DARA GmbH under Grants 50 TT 9536/50 TT 9733.

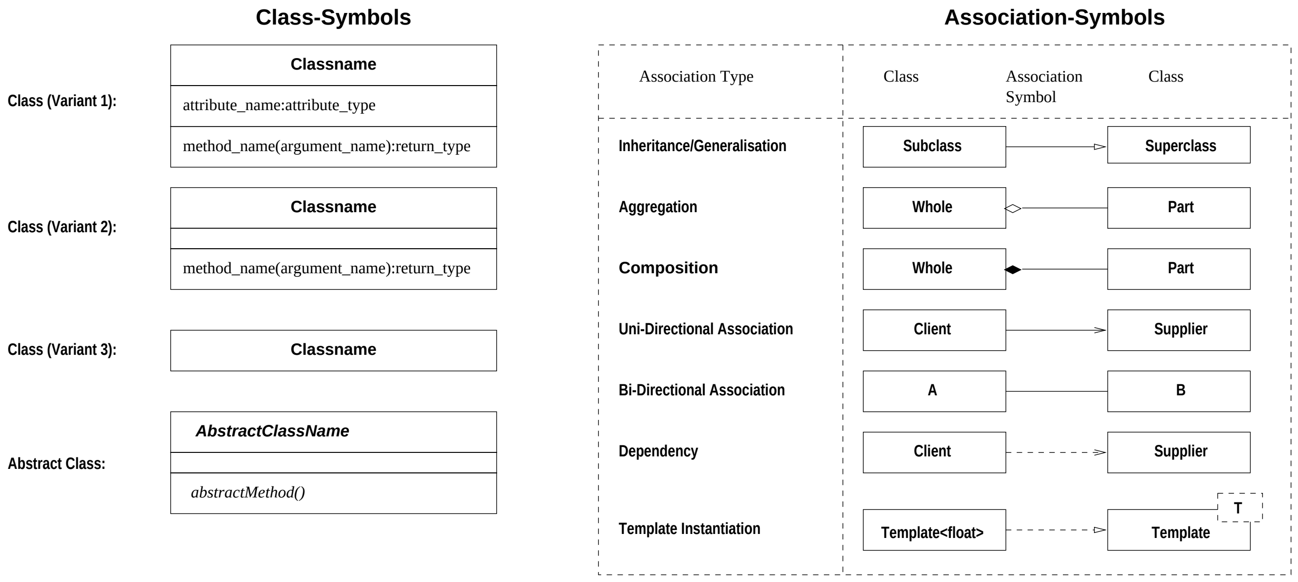

Figure 1: UML-Notation of class-symbols and association-symbols.

"Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships."

"Object-oriented design is a method of design comparsing the process of object-oriented decomposition and a notation for depicting the logical and physical as well as static and dynamic models of the system under design."

An object-oriented system consists of a set of objects, which are defined by different *classes*. The integral part of a class are methods and attributes. The attributes describe the data of an object, and the methods describe its behavior. To prevent an illegal access to internal attributes or methods of a class, an object-oriented language offers different levels of encapsulation. This means that e.g. a C++ class is able to contain *public*, *protected* and *private* elements.

The classes of an object-oriented system are associated by different types. The essential ones are (cf. also Figure 1):

- **Inheritance**

  Class Subclass contains all elements of class Superclass and some additional features. In this case it is advantageous for class Subclass to inherit all elements from class Superclass. Another aspect of inheritance is the association between an abstract class and its implementation. An abstract class contains only the signature of its methods, which declares name and parameters, but not the definition. Only objects of inherited classes, which have the abstract methods implemented can be instantiated.

- **Aggregation**

  Some attributes of class Whole are instances of class Part. In other words: Whole contains Part.

- **Uni-Directional Association**

  Class Client contains a reference to an object of class Supplier, but class Supplier does not know anything about class Client.

- **Bi-Directional Association**

  Class A and B contain references to each other.

- **Dependency**

  The methods of class Client use methods or attributes of class Supplier.

- **Template Instantiation**

  A class Template can be parameterized by the type parameter T. For example the class Template$<$float$>$ is an instantiation of Template with the type parameter float.

The design process of an object-oriented system starts by modeling a set of classes and their associations. A special notation of object-oriented modeling will ease the work of the designer.

The main notions, used in the context of object-oriented modeling can be found in e.g. in the UML-Glossary (Rat, 1997a).

2.2 The Unified Modeling Language

Neither a natural language nor a programming language is able to describe the modeling of a complex system in an easy way. For that reason G. Booch, I. Jacobson and J. Rumbaugh designed the Unified Modeling Language (UML). It is intended to include

- Model elements - fundamental modeling concepts and semantics,
- Notation - visual rendering of model elements,
- and Guidelines - idioms of usage within the trade (Rat, 1997b).

The UML notation provides a variety of diagrams for different purposes, like dynamic or static system views.

Figure 1 gives an overview of the UML notation, which is used in the following diagrams. The association types as described above are elements of *Class Diagrams*, that show the static structure of the model. Classes are represented as boxes, divided in up to three fields, whereas the upper field contains the class-name, the middle field contains the attribute names and the bottom field contains the method names. The attribute- and method fields may be omitted (cf. Variant 3 in fig. 1).

These diagrams contain the essential classes and their associations. Other kinds of UML-diagrams are *Use Case Diagrams* for the communication with users and orderers, *Collaboration- and Sequence Diagrams*, which describe the interaction between objects, *Activity Diagrams* for describing a specified operation and *State Diagrams* for depicting the states of a specified object.

2.3 Description schema of a Design Pattern

A design pattern describes a proposed solution of a permanent recurring software design problem. We use the description-schema of design patterns according to (Gamma et al., 1995) in this paper. A more detailed description of design patterns can be found in (Gamma et al., 1995, Sommerlad, 1996, Buschmann, 1996). The following list gives a short overview of the used description-schema:

Intent Describes the underlying problem and the basic principle of the design solution (cf. section 3.1).

Also Known As Other names of the pattern, if existing.

Motivation Depicts a scenario of a concrete design problem and gives a solution. In our case it describes the design of a point matching tool as an occasion for the design of a general matching tool (cf. section 3.2).

Applicability Itemizes the situations or conditions, in which the pattern can be applied (cf. section 3.3).

Structure Describes the static structure of classes, which are participating. As mentioned, we use the notation of the UML. The class structure, presented in the Motivation section, is a concrete instance of the more general class structure, presented in Structure-section (c.f. section 3.4).

Participants Lists the classes and objects, which are described by the design pattern (c.f. section 3.5).

Collaborations Describes the interactions between the objects to fulfill the task of the software (c.f. section 3.6).

Consequences Discusses the advantages and disadvantages of the pattern, and makes suggestions about variations (c.f. section 3.7).

Implementation Gives some implementation details, which may be language-dependent (c.f. section 3.8).

Sample Code A sample code, if necessary.

Known Uses Examples of applications, in which the design pattern is used (c.f. section 3.9).

Related Patterns Lists patterns, which describe a similar task, and shows the relations to other patterns (c.f. section 3.10).

The UML and Design Patterns are very helpful to improve the communication between developers, users and orderers. We use these aids to describe the design of the matching tool. The following section encloses the design pattern *Matching Tool*.

3 DESIGN PATTERN OF A MATCHING TOOL

The name of the Design Pattern is

Matching Tool

3.1 Intent

Create an 3D-description from a set of images, using appropriate matching techniques.

3.2 Motivation

A semiautomatic building extraction system system shall be extended with an automatic matching tool. The system is equipped with an object-oriented class interface, which contains classes of 2D/3D points and oriented images (Gülch and Müller, 1997). The matching tool shall create a 3D-description and implement several different matching techniques.

The basic idea is to model the 3D description as an object, which can be created from a set of oriented images. A matching algorithm has to be called when a client, a class that requests a service from another class, requires the 3D structure of this object.

We begin with the design of a point matching tool as a concrete instantiation of a general matching tool. In this case the 3D description is a 3D-point. The class MatchedPoint describes the attributes and methods of such a 3D-point.

The class MatchedPoint should fulfill the following requirements:

1. Clients, e.g. a slot or a callback function of a Graphical User Interface (GUI) in a semiautomatic system, are independent of the used matching technique, i.e. the change of the matching technique will not affect the client-class. This eases the redesign of the system, if a new matching technique is implemented. Additionally, clients can be reused within a context, in which some the used matching techniques are unknown.
2. Different matching techniques are exchangeable during runtime. The selection of the matching technique depends on
    - the image data at the reference position,
    - the knowledge about the 3D context,
    - or the users decision.
3. Similar matching techniques are represented by a refinement of a more general template class with one or more unbound type-parameters. This template describes on an abstract level a set of matching techniques.

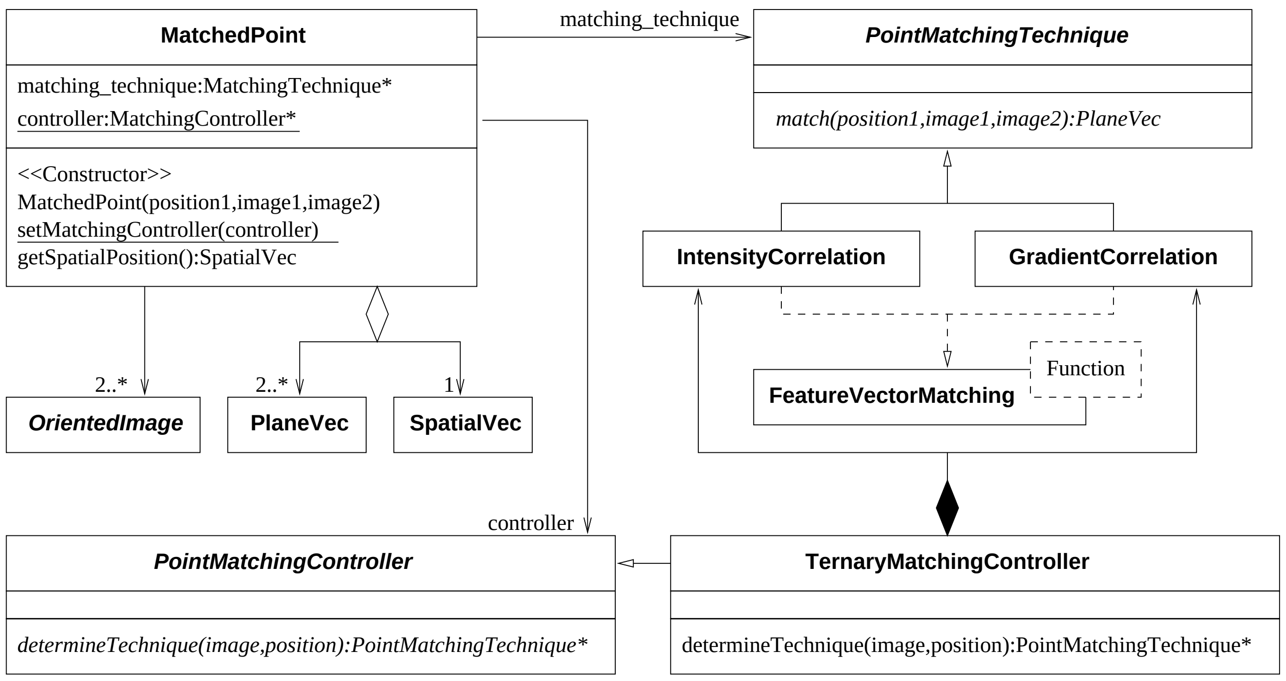
MatchedPoint
matching_technique:MatchingTechnique*
controller:MatchingController*
<<Constructor>>
MatchedPoint(position1,image1,image2)
setMatchingController(controller)
getSpatialPosition():SpatialVec
matching_technique
PointMatchingTechnique
match(position1,image1,image2):PlaneVec
IntensityCorrelation
GradientCorrelation
Function
FeatureVectorMatching
2..*
OrientedImage
2..*
PlaneVec
1
SpatialVec
controller
PointMatchingController
determineTechnique(image,position):PointMatchingTechnique*
TernaryMatchingController
determineTechnique(image,position):PointMatchingTechnique*

Figure 2: Class structure of the Point Matching Tool.

The class structure of the Point Matching Tool is shown in fig. 2. This structure is based on the Strategy Design Pattern (Gamma et al., 1995), which contains an abstract interface for a family of algorithms. In our case this family of algorithms consists of different matching techniques. The abstract interface, which contains at least one polymorph method, enables the exchangeability of the algorithm during runtime.

The class MatchedPoint is associated to the abstract class PointMatchingTechnique. This enables MatchedPoint to use the interface of PointMatchingTechnique for calling a matching algorithm without knowing any detail about the concrete matching technique. These are in the actual implementation of the Point Matching Tool the classes IntensityCorrelation and GradientCorrelation.

The class IntensityCorrelation performs a correlation of the grey-values in the image, while the class GradientCorrelation performs a correlation of the gradient. The algorithms of these matching techniques are very similar and differ only in some details. A common description of both algorithms and further ones is given by the template class FeatureVectorMatching.

The basic principle of Feature Vector Matching is to transform the image data of a local window, surrounding a given point, to a feature vector. The elements f_i of this feature vector are features, which are calculated from the image data of the local window. These features may be invariant under certain geometric or radiometric transformations.

Let the grey-value at position (r,c) in the local window be $g(r,c)$. The next examples show several kinds of feature vectors.

a) Simple grey-values:

$$f_i = g(r,c) \tag{1}$$

with

$$i = i(r,c) \tag{2}$$

b) Gradient, e.g. with the Sobel Operator:

$$f_i = \sqrt{g_r(r,c)^2 + g_c(r,c)^2} \tag{3}$$

c) Zernike moments A_{nl} as described in (Straub, 1991):

$$f_{2n+l} = A_{nl} \tag{4}$$

Generally the feature vector $\mathbf{f}$ can be expressed as a result of a transformation $\mathbf{T}$ from the image to the feature space:

$$\mathbf{f} = \mathbf{T}g(r,c) \tag{5}$$

After transformation to the feature space the correlation function between the feature vector of a reference image point $\mathbf{f}(r,c)$ and the one of a search image point $\mathbf{f}(r+\Delta r, c+\Delta c)$can be calculated.

$$\rho(r,c,r{+}\Delta r,c{+}\Delta c) = \frac{\sum_i^k (f_i^{(1)} - \bar{f^{(1)}})(f_i^{(2)} - \bar{f^{(2)}})}{\sqrt{\sum_i^k (f_i^{(1)} - \bar{f^{(1)}})^2 \sum_i^k (f_i^{(2)} - \bar{f^{(2)}})^2}} \tag{6}$$

with

$$\begin{aligned} f_i^{(1)} &= f_i(r,c) \\ f_i^{(2)} &= f_i(r+\Delta r, c+\Delta c) \end{aligned}$$

This correlation is calculated in a given search space of the search image. The search space results from an epipolar line, which is determined from the coordinate of the given point in the reference image. The point in the search space with maximum correlation is the corresponding point of the given point in the reference image.

The transformation of the image data to a feature vector is performed by a function object, which encapsulates a function in an object for use by other components (Musser and Saini, 1996), as a parameter of the FeatureVectorMatching-template. For each matching technique, based on Feature

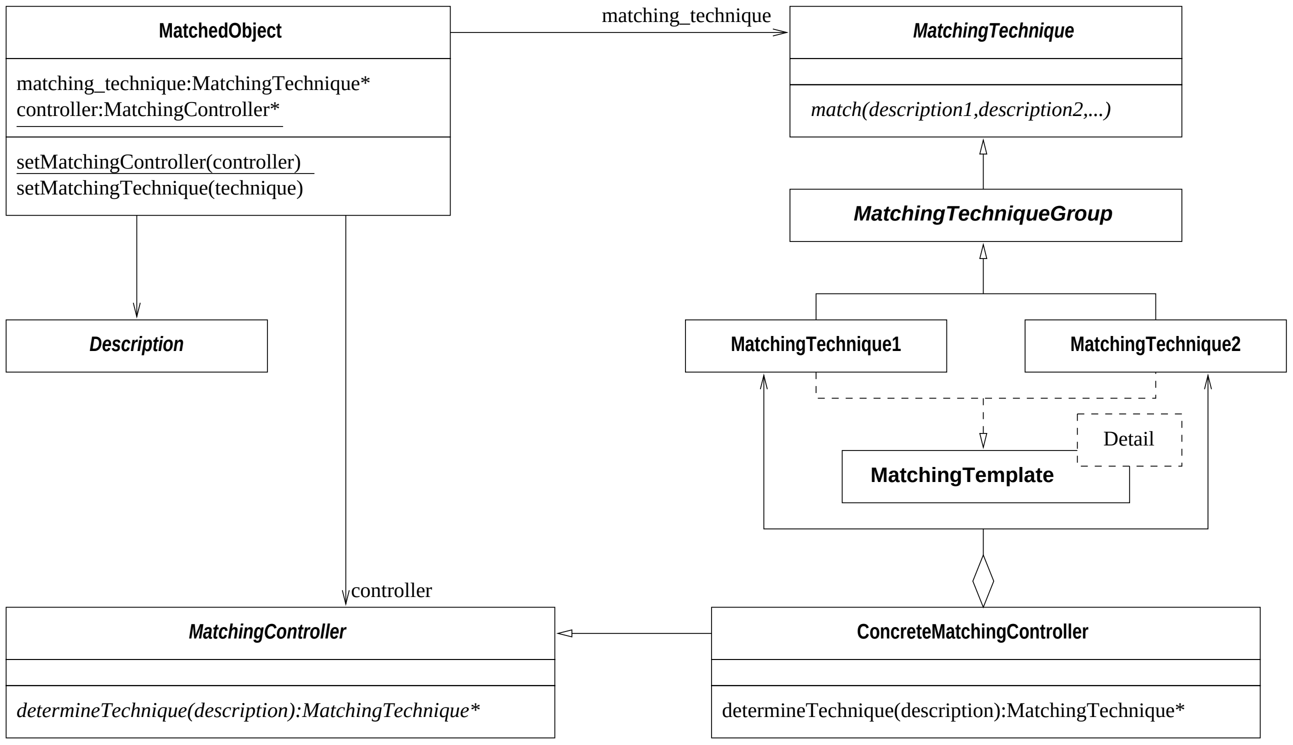
MatchedObject
matching_technique:MatchingTechnique*
controller:MatchingController*
setMatchingController(controller)
setMatchingTechnique(technique)
matching_technique
MatchingTechnique
match(description1,description2,...)
MatchingTechniqueGroup
MatchingTechnique1
MatchingTechnique2
Description
Detail
MatchingTemplate
controller
MatchingController
determineTechnique(description):MatchingTechnique*
ConcreteMatchingController
determineTechnique(description):MatchingTechnique*

Figure 3: Class structure of a matching tool

Vector Matching, such a function object has to be implemented.

All available objects with a PointMatchingTechnique-interface are implemented as attributes of the class TernaryMatchingController. This class is responsible for creation and removal of the objects of PointMatchingTechnique and provides them with additional parameters, e.g. the window-size. If the system will be extended with a further matching technique, only this class has to be changed. The method determineTechnique of this class comes to a decision about the best matching technique for a special position in an image. The classification into regions, point and edges, as described in (Förstner, 1994) can be used to determine the optimal matching technique. Tab. 1 describes the connection between this classification and a set of appropriate matching techniques, which are Feature-Based techniques, like Corner- and Line-based Matching, and Area-Based techniques, like Correlation- and Least Squares Matching.

Classification		Matching Technique
point	$\rightarrow$	Corner-based (Lang and Förstner, 1996) Correlation (Gradient,Intensity) Least Squares
edge	$\rightarrow$	Line-based Correlation (Intensity,Gradient) Least Squares
region	$\rightarrow$	Correlation (Intensity) Least Squares

Table 1: The connection between feature classification and matching technique.

The class TernaryMatchingController is a concrete Subclass of the abstract class PointMatchingController. It makes the determineTechnique-method available to classes, which are independent of the implemented matching techniques, like the class MatchedPoint. So this class is equipped with a reference to a object of PointMatching-Controller.

As it is not necessary, that several objects of MatchedPoint use several objects of PointMatchingController during the same time, the controller attribute of the class Matched-Point can be implemented as a class-scope attribute. A class-scope attribute refers for every instance to the same object, and it can be accessed without an instantiated object. Its implementation is similar to a global variable in a procedural system.

3.3 Applicability

The matching tool can be used in the following cases:

- Several matching techniques are used within one system.
- A template of similar matching techniques is needed.
- Objects, which are created by a matching technique, are designated to be independent of the implemented technique.

3.4 Structure

The structure of the matching tool is shown in fig. 3.

3.5 Participants

Description (OrientedImage, PlaneVec, SpatialVec)

A set of images or higher level descriptions and additional knowledge, which contains all the necessary information to create the desired object.

MatchedObject (MatchedPoint)

The object, whose attributes are determined by a matching process. For topographic applications, this may be a topographic object.

MatchingTechnique (PointMatchingTechnique)

An abstract interface for a matching technique.

MatchingTechniqueGroup (AreaBasedMatching, FeatureBasedMatching)

This class represents an intermediate level of abstraction. A group of matching techniques with common attributes and methods is described by this class.

MatchingTechnique1, MatchingTechnique2 (IntensityCorrelation, GradientCorrelation)

Concrete implementations of matching techniques.

MatchingTemplate (FeatureVectorMatching)

A template, which describes the community of several matching techniques. This class handles in contrast to MatchingTechniqueGroup with internal or algorithmic communities.

MatchingController (PointMatchingController)

An abstract interface, which is accessed by the MatchedObject to determine an appropriate matching technique.

ConcreteMatchingController (TernaryMatchingController)

The concrete implementation of MatchingController, which has the knowledge about all available matching techniques.

3.6 Collaborations

If a client requires parameters, resulting from a matching algorithm, the instance of MatchedObject calls the match-method of the object of class MatchingTechnique. If a client needs only parameters, which need no matching technique for determination, e.g. the 2D-Coordinate of a point in the reference image, the matching algorithm will not be called.

If no matching technique is associated to an instance of MatchedObject, it asks the controller-object to determine the matching technique. Alternatively a client can set the matching technique explicitly.

The object of the Description-classes are used by the object of class MatchingTechnique to get the image data or higher level descriptions and additional knowledge based information, e.g. orientation data and a reference position.

An instance of class ConcreteMatchingController creates the objects of several implemented Matching Techniques. It provides these objects with matching technique specific parameters and may use some of the description data from an instance of class MatchedObject to determine the matching technique.

3.7 Consequences

The matching tool pattern has the following advantages and disadvantages.

+ The class MatchedObject is independent of the concrete matching technique and the client does not need an explicit call of the matching algorithm.

+ The class MatchingTemplate enables to implement communities of several matching techniques only once.

+ If an new matching technique is implemented, only the class ConcreteMatchingController has to be changed.

- Before the first instance of MatchedObject is created, the class-scope method setMatchingController, which is callable without an instance of MatchedObject, has to be called. This method-call increases the initialization part of the surrounding system.

3.8 Implementation

The generic programming technique (Musser and Saini, 1996), which is used to design the class MatchingTemplate, causes sometimes compile problems. Most c++-compilers differ in the way of interpreting templates (cf. slide-in in (Eisenecker, 1996)).

The class MatchedObject may be inherited from a more general object description. In this case the programmer has to check the consistency of the new matching features with the methods of the superclass interface.

An alternative of inheritance is automatic type conversion to another description of the MatchedObject. This is recommended for primitive objects, like points or edges.

3.9 Known Uses

The Design Pattern “Matching Tool” is used for designing and integrating a point matching tool in a Semiautomatic Building Extraction System as mentioned above. Its purpose is to reduce the number of interactive actions. The point matching tool is used to measure homologue points

1. at roof-edges to determine the 3D-position of the building primitive along the projection line of the image

2. and nearby buildings, whose bottom height cannot interactively measured, because the ground edges are occluded by vegetation.

Actually, the matching techniques IntensityCorrelation and GradientCorrelation are implemented (cf. 3.2). The two matching techniques were tested on several types of points (cf. tab. 2). The decisions about successful matching were made interactively. The points for both techniques were approximately chosen at the same position.

The test results show, that Gradient Correlation is a little more successful at gable point positions. This can be explained thereby, that the intensity values nearby gable points are stronger changed by a projective transformation, than the gradient values. This fact indicates the dependency of the optimal matching technique on the local image

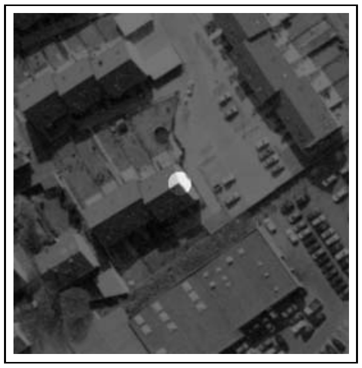

Figure 4: Left image with given point (e.g. gable point).

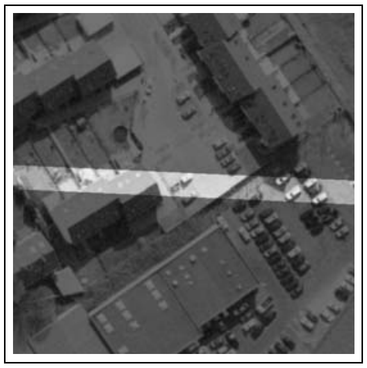

Figure 5: Right image with epipolar search space.

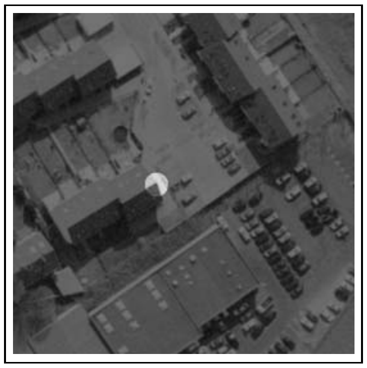

Figure 6: Right image with correctly matched point.

Point location	Technique	Points	Matches	
Gable point	Intensity Cor.	41	26	(63%)
Roof Top	Intensity Cor.	22	22	(100%)
Street	Intensity Cor.	26	24	(92%)
Gable point	Gradient Cor.	43	29	(67%)
Roof Top	Gradient Cor.	22	22	(100%)
Street	Gradient Cor.	25	18	(72%)

Table 2: Test results of the point matching tool with Intensity and Gradient Correlation (Street positions at urban streets, mainly without markings).

features. All points on roof tops were successfully matched at all examined positions by both methods. At locations in the street, the intensity correlation is better, which is explained by the lack of strong edges.

Fig. 4-6 show an example of point matching in the semiautomatic building extraction system. In this case the method determineTechnique has found a GradientCorrelation as the best technique for this point, which has successfully been matched by the automatically chosen technique.

3.10 Related Patterns

- Strategy (Gamma et al., 1995)

  The Matching Tool pattern can be understood as a refinement of the Strategy Pattern specialized to matching techniques.

- Generic techniques of the STL (Musser and Saini, 1996)

  The generic techniques, which are used by the STL, are very helpful for designing the class MatchingTemplate.

4 CONCLUSIONS

In this paper we have shown an object-oriented design of matching techniques. Beginning with the task to design a point matching tool, we found a general design pattern for matching techniques. This design pattern offers the possibility to create complex objects directly from low level descriptions like images. E.g. a 3D-point object can be created from only two images and a reference position in one image. Details of the implemented matching techniques are hidden, and they can be varied with no effect to other software components. The matching technique can be selected automatically, using local features like edges, points or regions. The implementation of this design pattern demonstrates, that object-oriented modeling techniques also can be applied on algorithmic models, like matching techniques.

One of our future tasks is to apply the presented design pattern to other topographic objects and to extend the variety of available matching techniques. For this intention we need to model a set of further matching techniques, which is actually not worked out. We will integrate these new components into a semiautomatic building extraction system to reduce further the amount of interactive operations.

ACKNOWLEDGMENTS

The inspiration and support by Prof. Dr.-Ing. Wolfgang Förstner and Dr.-Ing. Eberhard Gülch are gratefully acknowledged.

REFERENCES

Booch, G., 1994. Object-oriented Analysis and Design. With Applications. Benjamin/Cummings Publishing Company, Inc.

Buschmann, F., 1996. Wie beschreibt man entwurfsmuster? OBJEKTspektrum.

Eisenecker, U., 1996. Generatives programmieren in c++. OBJEKTspektrum.

Englert, R. and Gülch, E., 1996. One-eye stereo system for the acquisition of complex 3D building descriptions. GIS.

Förstner, W., 1986. A feature based correspondence algorithm for image matching. In: International Archives of Photogrammetry and Remote Sensing, Vol.26-3/3, Rovaniemi, pp. 150–166.

Förstner, W., 1994. A framework for low level feature extraction. In: J.-O. Eklundh (ed.), Computer Vision - ECCV ’94, Vol. II, Lecture Notes in Computer Science, 801, Springer-Verlag, pp. 383–394.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. Design Patterns. Addison-Wesley.

Gülch, E., 1997. Application of semi-automatic building acquisition. In: A. Grün (ed.), Automatic Extraction of Man-Made Objects from Aerial and Space Images (II), Birkhäuser, Basel.

Gülch, E. and Müller, H., 1997. Object-oriented software design in semiautomatic building extraction. In: Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision III, SPIE Proceedings, Vol. 3072.

Heipke, C., Mayr, W., Wiedemannn, C. and Ebner, H., 1997. Automatic aerotriangulation with frame and three-line imagery. In: Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision III, SPIE Proceedings, Vol. 3072.

Lang, F. and Förstner, W., 1996. Surface reconstruction of man-made objects using polymorphic mid-level features and generic scene knowledge. In: ISPRS Congress, Vienna.

Lang, F. and Schickler, W., 1993. Semiautomatische 3D-Gebäudeerfassung aus digitalen Bildern. Zeitschrift für Photogrammetrie und Fernerkundung 5, pp. 193–200.

Musser, D. R. and Saini, A., 1996. STL Tutorial and Reference Guide. Addison-Wesley.

Rat, 1997a. UML Semantics Glossary. Version 1.0 edn. URL: `http://www.rational.com`.

Rat, 1997b. Unified Modeling Language. Version 1.0 edn. URL: `http://www.rational.com`.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., 1991. Object-Oriented Modeling and Design. Prentice-Hall, Inc.

Sommerlad, P., 1996. Entwurfsmuster für software-architektur. OBJEKTspektrum.

Straub, B., 1991. Ein Verfahren zur Rekonstruktion von dreidimensionalen Objektmodellen aus digitalen Bilddaten. PhD thesis, Universität Stuttgart.